| ***Computer Engineering Department*** |
| --- |
| ***CE100L: Computing Fundamentals & Programming*** |

| ***Course Instructor: Usama Bin Shakeel*** | ***Dated: 07/01/2022*** |
| --- | --- |
| ***Teaching Assistant: Aqsa Khalid*** | ***Semester: Fall 2021*** |
| ***Lab Engineer: Nadir Abbas*** | ***Batch: BSCE2021*** |

# **Lab 11B. Loops and File Handling in C++**

| **Name** | **Roll number** | **Report**  **(out of 100)** | **Scaled to 10** | **Total**  **(out of 10)** |
| --- | --- | --- | --- | --- |
| Muhammad Abubakar Saif | BSCE21017 |  |  |  |

Checked on: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## **Objective**

The objective of this lab is to perform lab tasks in multiple steps using the concept of loop and file handling.

## **Equipment and Component**

| **Component Description** | **Value** | **Quantity** |
| --- | --- | --- |
| Computer | Available in lab | 1 |

## **Conduct of Lab**

1. Students are required to perform this experiment individually.
2. In case the lab experiment is not understood, the students are advised to seek help from the course instructor, lab engineers, assigned teaching assistants (TA) and lab attendants.

## **Theory and Background**

A statement or a set of statements that is executed repeatedly is called a loop. The statement(s) in a loop are executed for a specified number of times or until some given condition remains true. In C++, there are three kinds of loop statements. These are: The “for” loop. The “while” loop. The “do-while” loop.

The “while” loop:- It is a conditional loop statement. It is used to execute a statement or a set of statements as long as the given condition remains true.

In for loop the number of iterations to be done is already known. In other words it is used to execute a statement or a set of statements for a fixed number of times.

**File Handling:**

In all programs we have written so far the input was given from the keyboard and output was received on the computer screen. This method works well when inputs are not too many and are small. But when the input becomes large it is not possible to enter data from the keyboard. If there is a mistake in entering data all data will have to be entered again. Similarly if the output is large it is necessary to keep it permanently stored on the disk for future use.

The data files provide a solution to this problem. The files stored on disk are used to give input into the program. The input data is written into the program. The program reads the data from the file. Similarly the program writes the output in a file on the disk. The output written in the file can then be displayed and/or printed on the printer. File Handling in C++ is quite different from that in procedural languages. C++ provides an easy way of File Handling. In procedural languages files are processed on the disk by using functions but in C++ objects are used for the same purpose.

**Streams and Files**

In C++ special classes known as stream classes are used to handle data streams. These classes are called stream classes. Stream classes are defined in header files and these are included in the program through their header files. The objects of stream classes are used to control flow of data.

For example, in C++ programs we commonly include the header file “iostream”. This header file contains the objects “cin”, “cout”, “cerr”, and “clog”. These objects are used to move streams of data across various parts of the computer. Following are the most commonly used C++ streams classes

**ofstream:** Stream class to write on files

**ifstream:** Stream class to read from files

**fstream:** Stream class to both read and write from/to files.

These classes are derived directly or indirectly from the classes istream and ostream. We have already used objects whose types were these classes: cin is an object of class istream and cout is an object of class ostream. Therefore, we have already been using classes that are related to our file streams. And in fact, we can use our file streams the same way we already know how to use cin and cout, with the only difference that we have to associate these streams with physical files.

Let’s see an example:

**// basic file operations**

#include

#include

using namespace std;

int main ()

{

ofstream myfile;

myfile.open (“example.txt”);

myfile <<”Writing this to a file.\n”;

myfile.close();

return 0;

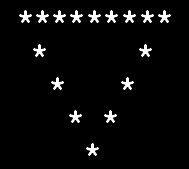
}

This code creates a file called **example.txt** and inserts a sentence into it in the same way we are used to with **cout**, but using the file stream **myfile** instead.

**Lab Tasks**

1. Write a C++ code for v shape and flipped v shape using the concept of nested loops and file handling in multiple steps given below:

* Take input from a file
* Take input from a user
* Pass this information to a function to print v shapes
* Write these shapes to a file

****

Main function for both Tasks:

**int** main() {

**int** dim = 0, choice;

**char** type;

again:

cout << **"Tell by which mean you want to input values to draw shape: "** << endl;

cout << **"Enter 1 for reading from file."** << endl;

cout << **"Enter 2 to give input directly."** << endl;

cin >> choice;

**if** (choice > 0 && choice < 3) {

**switch** (choice) {

**case** 1:

readFromFile(dim, type);

**break**;

**case** 2:

cout << **"Enter dimension of the shape: "**;

cin >> dim;

cout << **"Enter the character: "**;

cin >> type;

**break**;

}

} **else** {

cout << **"Invalid Selection, Try again"** << endl;

**goto** again;

}

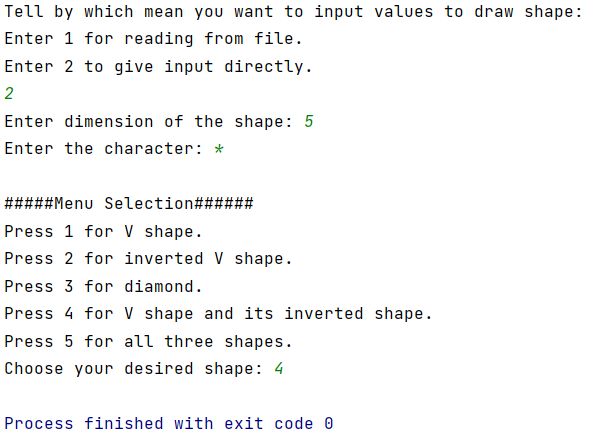
shapeMenu(dim,type);

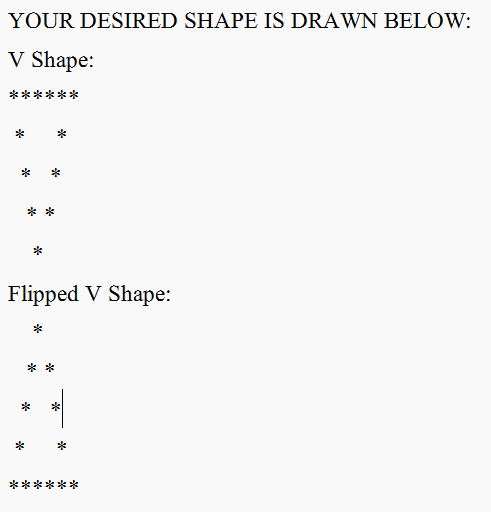
**return** 0;

}

Task 1:

| **void** flipVShape(**int** dim, **char** letter) {  ofstream draw;  draw.open(**"Shape.txt"**,ios :: app);  draw<<**"Flipped V Shape: "**<<endl;  **int** space = 1;  **int** n = 1;  **for** (**int** i = 1; i <= dim; i++) { *// makes new rows*  **for** (space = i; space < dim; space++) { *//creates spaces in a row*  draw << **" "**;  }  **for** (space = 1; space == 1 && (i != dim || dim == 2) ; space++) { *//prints first letter after created spaces in the same row*  draw << letter;  }  **for** (**int** mid = 1; mid <= n && i != 1 && (i != dim || dim == 2) ; mid++) { *//creates spaces between first and last character*  draw << **" "**; *//of the row except for first and last row*  }  **if** (i != 1 && (i != dim || dim == 2)) { *//print last character of the rows except for first and last row*  draw << letter;  }  **for** (**int** last = 1; last <= dim+1 && i == dim && dim != 2; last++) { *//prints letters in the last row*  draw << letter;  }  draw << endl; *//jump to new line*  n = n + 2;  **if** (i<2 && i>0){ *//runs only once during this function to reset the value of n to 1 in case of i = 1*  n = 1; *// for proper functioning of this code*  }  *//cout<<n<<endl;*  }  *//cout<<n;*  draw.close();  }  **void** vShape(**int** dim, **char** letter){  ofstream draw;  draw.open(**"shape.txt"**,ios::app);  draw<<**"V Shape: "**<<endl;  **int** space = 1,n;  **for** (**int** i = 1; i <= dim; i++) { *// makes new rows*  **for** (**int** start = 1; start <= dim && i == 1 && dim != 2; start++) { *//prints letters in the first row*  draw << letter;  }  **for** (space = 1; space < i; space++) { *//creates spaces in a row*  draw << **" "**;  }  **for** (space = 1; space == 1 ; space++) { *//prints first letter after created spaces in the same row*  draw << letter;  }  **for** (**int** mid = 1; mid <= n && i != 1 && i != dim ; mid++) { *//creates spaces between first and last character*  draw << **" "**; *//of the row except for first and last row*  }  **if** (i != 1 && (i != dim || dim == 2)) { *//print last character of the rows except for first and last row*  draw << letter;  }  draw << endl; *//jump to new line*  n-=2;  **if** (i<2 && i>0){ *//runs only once during this function to reset the value of n to 1 in case of i = 1*  n = (dim\*2)-5; *// for proper functioning of this code*  }  }  draw.close();  } |
| --- |

Output:

**(Shape.txt):  
**

1. Write a C++ code for a hollow shaped diamond using the concept of nested loops and file handling in multiple steps given below:

* Take input from a file
* Take input from a user
* Pass this information to a function to print diamond
* Write a diamond to a file

#### 

| **void** drawDiamond(**int** dim, **char** letter){  ofstream draw;  draw.open(**"Shape.txt"**, ios :: app);  draw<<**"Diamond Shape: "**<<endl;  **int** space = 1;  **int** n = 1;  **for** (**int** i = 1; i <= dim/2 ; i++) { *// makes new rows*  **for** (space = i; space < dim; space++) { *//creates spaces in a row*  draw << **" "**;  }  **for** (space = 1; space == 1 ; space++) { *//prints first letter after created spaces in the same row*  draw << letter;  }  **for** (**int** mid = 1; mid <= n && i != 1 ; mid++) { *//creates spaces between first and last character*  draw << **" "**;  }  **if** (i != 1) { *//print last character of the rows*  draw << letter;  }  draw << endl; *//jump to new line*  n = n + 2;  **if** (i<2 && i>0){ *//runs only once during this function to reset the value of n to 1 in case of i = 1*  n = 1; *// for proper functioning of this code*  }  }  **for** (**int** i = (dim/2)+1; i <= dim; i++) { *// makes new rows after mid*  **for** (space = 1; space < i; space++) { *//creates spaces in a row*  draw << **" "**;  }  **for** (space = 1; space == 1 ; space++) { *//prints first letter after created spaces in the same row*  draw << letter;  }  **for** (**int** mid = 1; mid <= n && i != dim ; mid++) { *//creates spaces between first and last character*  draw << **" "**; *//of the row except for last row*  }  **if** (i != dim) { *//print last character of the rows except last row*  draw << letter;  }  draw << endl; *//jump to new line*  n-=2;  **if** (i<2 && i>0){ *//runs only once during this function to reset the value of n to 1 in case of i = 1*  n = (dim\*2)-5; *// for proper functioning of this code*  }  }  draw.close();  } |
| --- |

Helper Function (Menu Selection):

**void** shapeMenu(**int** dim, **char** letter){

ofstream clear;

clear.open(**"Shape.txt"**);

clear<<**"YOUR DESIRED SHAPE IS DRAWN BELOW: "**<<endl;

**int** user;

again:

cout<<endl;

cout<<**"#####Menu Selection######"**<<endl;

cout<<**"Press 1 for V shape."**<<endl;

cout<<**"Press 2 for inverted V shape."**<<endl;

cout<<**"Press 3 for diamond."**<<endl;

cout<<**"Press 4 for V shape and its inverted shape."**<<endl;

cout<<**"Press 5 for all three shapes."**<<endl;

cout<<**"Choose your desired shape: "**;

cin>>user;

*//if (choice > 0 && choice < 6) {*

**switch** (user) {

**case** 1:

vShape(dim,letter);

**break**;

**case** 2:

flipVShape(dim,letter);

**break**;

**case** 3:

drawDiamond(dim,letter);

**break**;

**case** 4:

vShape(dim,letter);

flipVShape(dim,letter);

**break**;

**case** 5:

vShape(dim,letter);

flipVShape(dim,letter);

drawDiamond(dim,letter);

**break**;

**default**:

cout << **"Invalid Input, Try again"** << endl;

**goto** again;

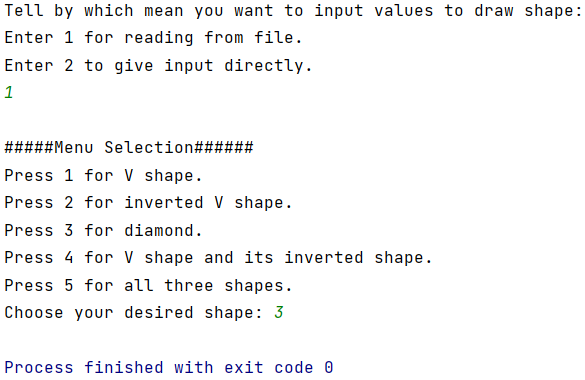
**break**;

}

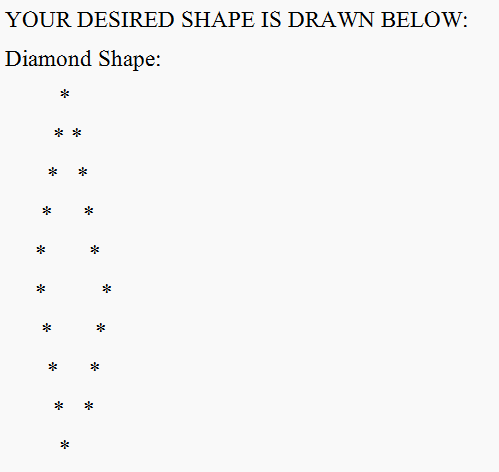
clear.close();

}

**Output (Task 2):**



**(Shape.txt):**



#### **Assessment Rubric for Lab**

**Method for assessment:**

Lab reports and instructor observation during lab sessions. Outcome assessed:

a. Ability to conduct experiments, as well as to analyze and interpret data (P) b. Ability to function on multi-disciplinary teams (A)

c. Ability to use the techniques, skills, and modern engineering tools necessary for engineering practice (P)

| Performance metric | Mapping (task no. and description) | | Max marks | Exceeds expectation | Meets expectation | Does not meet expectation | Obtained marks |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1. Realization of experiment (a) | 1 | Functionality | 40 | Executes without errors excellent user prompts, good use of symbols, spacing in output. Through testing has been completed (35-40) | Executes without errors, user prompts are understandable, minimum use of symbols or spacing in output. Some testing has been completed (20-34) | Does not execute due to syntax errors, runtime errors, user prompts are misleading or non-existent. No testing has been completed (0-19) |  |
| 2. Teamwork (b) | 1 | Group Performance | 5 | Actively engages and cooperates with other group member(s) in effective manner (4-5) | Cooperates with other group member(s) in a reasonable manner but conduct can be improved (2-3) | Distracts or discourages other group members from conducting the experiment (0-1) |  |
| 3. Conducting experiment (a, c) | 1 | On Spot Changes | 10 | Able to make changes (8-10) | Partially able to make changes (5-7) | Unable to make changes (0-4) |  |
| 2 | Viva | 10 | Answered all questions (8-10) | Few incorrect answers (5-7) | Unable to answer all questions (0-4) |  |
| 4. Laboratory safety and disciplinary rules (a) | 1 | Code commenting | 5 | Observes lab safety rules; handles the equipment and parts with care and adheres to the lab disciplinary guidelines aptly (4-5) | Generally observes safety rules and disciplinary guidelines with minor lapses (2-3) | Disregards lab safety and disciplinary rules (0-1) |  |
| 5. Data collection (c) | 1 | Code Structure | 5 | Excellent use of white space, creatively organized work, excellent use of variables and constants, correct identifiers for constants, No line-wrap (4-5) | Includes name, and assignment, white space makes the program fairly easy to read. Title, organized work, good use of variables (2-3) | Poor use of white space (indentation, blank lines) making code hard to read, disorganized and messy (0-1) |  |
| 6. Data analysis (a, c) | 1 | Algorithm | 20 | Solution is efficient, easy to understand, and maintain (15-20) | A logical solution that is easy to follow but it is not the most efficient (6-14) | A difficult and inefficient solution (0-5) |  |
| 7. Computer use (c) | 1 | Documentation | 5 | Timely documented (4-5) | Late documented (2-3) | Not documented (0-1) |  |
|  | Max Marks (total): | | 100 | Obtained Marks (total): | | |  |

Lab Engineer Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_